Improved Interconnect Sharing by Identity Operation Insertion

Dirk Herrmann Rolf Ernst

Institut für Datenverarbeitungsanlagen
Technische Universität Braunschweig, Germany

Abstract

This paper presents an approach to reduce interconnect cost by insertion of identity operations in a CDFG. Other than previous approaches, it is based on systematic pattern analysis and automated transformation selection. The cost function controlling transformation selection is derived with statistical experiments and is optimized using practical benchmarks. The results show significantly reduced interconnect cost for most register architectures and application examples.

1 Introduction

In high-level synthesis, *allocation* determines the mapping of operations to functional units (*FUs*) and values to registers and thus defines the resulting interconnect structure. One objective of allocation is to minimize cost of interconnect, i.e., of wires, (bus) drivers and multiplexers. The potential of interconnect sharing depends on the frequency of similar patterns of operations and data transfers in the control and data flow graph (*CDFG*). Figure 1a gives an example for matching computation patterns that can use the same connections. Especially the cluster oriented allocation approaches [5], [7], [2] try to exploit this observation. However, none of the approaches mentioned above performs transformations on the CDFG to extend the set of similar patterns.

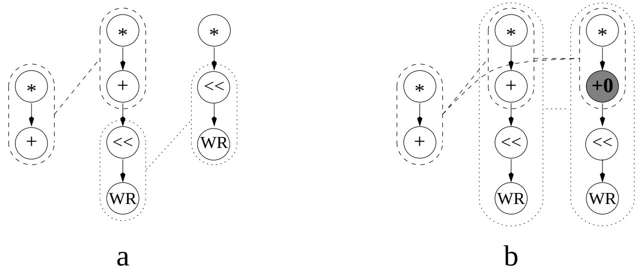

Figure 1: Aligning Computation Patterns

As a first approach to pattern adaptation, [6] presents a CDFG transformation technique, which extends the solution space for the allocation problem. By inserting identity operations (*deflection operations*) into the CDFG, previously unrelated patterns of operations and data transfers can be aligned, thus increasing the number of similar patterns. The result of such a transformation can be seen in figure 1b: The shaded zero-add operation has been inserted, thus eliminating the need for a connection between the multiplier and the shifter. Although based on a local search algorithm that only considers one identity operation at a time and depends on manual interaction, results in [6] show a large potential for interconnect savings. Still, the criteria for selecting transformations which are most likely to reduce cost in allocation are only roughly sketched. Further, this approach is specifically targeted towards a special (the *dedicated*) register file model.

Trading additional identity operations for speed or reduced interconnect cost is a common technique in computer architecture and compiler design. Load-store architectures, e. g., require the compiler to insert explicit instructions for memory read and write transfers. This simplifies instruction set, memory interface and reduces memory traffic at the cost of few additional instructions. To eliminate the need for register-register connections and dedicated register move instructions, data transfers between registers are often realized using addition of zeros (`R1 = R2 + 0`). For high-level synthesis, algebraic transformations that generate redundant operations are applied in [4] and [3] to reduce the critical path or interconnect cost.

We present a generalized approach to the alignment of computation patterns by insertion of identity operations. It differs from previous solutions in several aspects: Several transformations are performed in a single step which allows to reduce the number of transform/evaluate cycles. Further, the approach is independent of the register file implementation. Finally, our cost function is based on the observation that with increasing length of matching computation patterns, interconnect sharing is likely to improve. Therefore, our approach favors transformations that generate longer matches. It will be discussed, how possible transformations are detected and how the complexity of this step can be reduced. The incorporation of several insertions into a conflict free transformation with high probability for reduced interconnect cost will be described. The presented formal criteria for detection and selection of transformations are easily combined with other synthesis steps to a fully automated synthesis.

The paper is organized as follows: First, we will introduce the definitions and terminology used throughout the paper. The problem of determining a promising set of transformations is formulated and analyzed. Then, an algorithm will be outlined, followed by a set of experiments. Finally we summarize our results.

2 Definitions and Terminology

Definition 1: A computation *pattern* is a sequence $op_1, e_{1,2}, op_2, \ldots, op_{n-1}, e_{n-1,n}, op_n$ of operations and edges that form a directed non circular path in the CDFG.

Figure 2 gives an example of a CDFG and the corresponding set of patterns. In the following we assume that the hardware module selection for the operations of the CDFG is already performed.

This work is supported by the Deutsche Forschungsgemeinschaft (DFG)

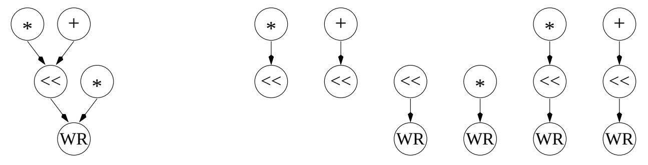

Figure 2: Patterns in a CDFG

Thus, every edge corresponds to a connection between dedicated ports of the FUs assigned to the source and target operation, while a pattern corresponds to a *path* of connections. The fact that two edges connect corresponding ports of FUs of the same type is an indication for a potential interconnect sharing between these edges and is the motivation for the next definition.

Definition 2: Two patterns P_1 and P_2 of the same length *match* if the following two conditions hold: First, corresponding operations use the same type of FUs and, second, corresponding edges will connect to the same ports of those FUs.

Since the detection of matching patterns is a central operation in our approach it has to be performed very fast. Thus we define patterns to be sequences of connections rather than subgraphs, because the matching problem between subgraphs is closely related to the subgraph isomorphism problem, which is NP-complete [1].

Whether or not an operation has a corresponding identity operation depends in some cases on the type of FU which is assigned to that operation. An increment operation may be executed on an ALU as well as on a dedicated incrementer. Only when assigned to an ALU, the increment operation has a corresponding identity operation. On some FUs different identity operations have to be distinguished, e. g. on a multiplier: $id_{MUL,1}(a) = a * 1$ and $id_{MUL,2}(a) = 1 * a$.

This defines a set of identity functions $id_{f,if}(a) = a$, where $f \in$ FU and id identity function of f. Sequences as store/load or increment/decrement are not considered as identity functions, here.

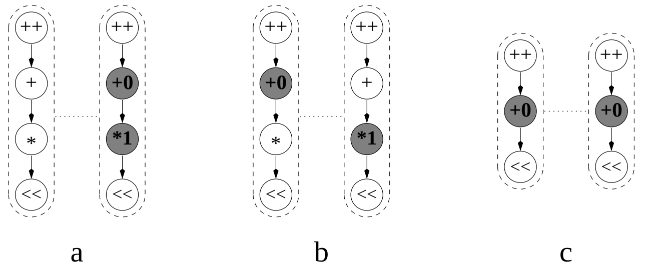

Figure 3: Different Types of Insertions

The basic transformation step is replacing an edge of the CDFG by one or more identity operations. The objective of such a transformation is that the resulting computation pattern matches some other pattern of the CDFG (figure 3a).

Definition 3: An *insertion* is a pair e, S, where e is an edge of the original CDFG and S is a sequence $id_1, id_2, \ldots, id_n$ of identity operations. A set I of insertions is *compatible* if for every edge e of the CDFG there is at most one insertion for e in I.

Figure 3b shows, that some matches can not be achieved by single insertions: Both, the multiplication by one as well as the addition of zero have to be inserted in order to achieve an additional pattern match. Thus, in the general case, sets of insertions have to be applied in combination. On the other hand, situations like in figure 3c are to be avoided, since identity operations at corresponding positions of a match don't improve interconnect sharing but may increase hardware requirements.

Definition 4: A compatible set I of insertions is called an *alignment* if there is at least one set P of at least two patterns in the original CDFG which do not match, but for which after application of I the resulting patterns P' match. A set of alignments is *compatible* if the corresponding union set of insertions is compatible. An alignment is called *minimal* if it does not insert identity operations which only match against other identity operations as in figure 3c.

3 Problem Formulation

The general objective of our approach is to determine a set of insertions I, such that the number of connections in the resulting circuit is minimized. The set of possible insertions is infinite since for each edge a sequence of identity operations of arbitrary length can be inserted. Thus we restrict the search space to compatible sets of minimal alignments, which is finite but still huge. Now, the problem addressed in this paper can be stated as follows:

Determine a compatible set of minimal alignments A, such that

- the resulting set of pattern matches maximizes the potential for interconnect sharing while
- no other synthesis constraints are violated.

The effect on performance and interconnect area when inserting additional operations into a CDFG can not be accurately predicted without performing a full synthesis. Consequently, a heuristic approach to determine a promising set of insertions is required that, firstly, limits the set of insertions that have to be considered and, secondly, estimates effects of insertions without need for a full synthesis.

4 Problem Analysis

	nodes	edges	patt.	align.
binary search	18	26	143	56
median filter	21	44	129	18
bubble sort	27	34	131	32
FIR filter	42	48	260	186
blue screen	44	72	317	718

Table 1: Pattern and Insertion Counts

Table 1 gives an impression of the number of patterns and alignments for a set of benchmarks when using the three FU types ALU, multiplier and shifter. We only counted alignments which only contain a single insertion, i. e. an insertion which aligns two patterns without depending on other insertions to be performed as well. Constellations as in figure 3b were not considered. Due to this fact and since most sets of alignments can form an alignment themselves the actual number of alignments is orders of magnitude larger.

Due to the theoretical and practical complexity it is infeasible to consider all possible sets of alignments. Heuristics like critical path length or variable lifetimes can be used to exclude edges from the set of possible insertions as has been proposed in [6]. Although this solution offers good results in common cases it does not limit the theoretical complexity of the problem. In contrast, by limiting the length of patterns that are to be considered, the number of patterns as well as the number of insertions and alignments can be reduced. This way, since not all possible alignments are considered, optimization potential can be traded for execution time of the optimization process.

In the following we will discuss how restricting the search space to shorter patterns influences the optimization potential. The results will be used to derive a cost function to estimate the improvements on interconnect sharing potential for a given set of potential alignments.

4.1 Pattern Length Limitation and Cost Function

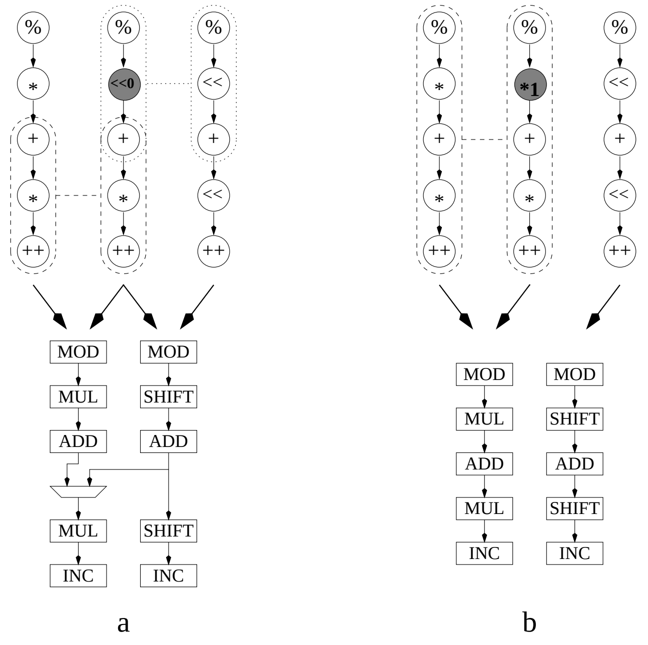

Figure 4: Length of Pattern Matches

Figure 4 shows a situation, where two different insertions can be applied to eliminate a connection between a modulo unit and an adder. Assume that two FUs of each type are provided (for simplicity registers between FUs are not shown). Solution a) requires one additional connection, since the middle pattern only partly matches each of the others. Examining the patterns of the CDFG, we see that for both solutions the number of matching patterns of length 2 is equal. Solution a) has two matches of length 3, whereas solution b) has three of them, plus two matches of length 4 and one match of length 5.

The example suggests that effects on longer patterns should be considered to rate transformations. A statistical experiment on a large set of pseudorandom CDFGs [8] was conducted to investigate, how much influence matches of patterns of different lengths have on the resulting interconnect cost. The results showed that with increasing pattern length the relative influence of the number of matches decreases and that using a maximum pattern length of four would be sufficiently accurate. Further we observed that for this limited range of pattern lengths the average number of saved connections has a close to linear dependency on the number of matching patterns of the different lengths, which allows to use a linear cost function

$$c = \sum_{l=2}^{4} a_l f(\mathrm{cdfg}, l)$$

where only patterns with lengths from 2 to 4 are considered and a_l is the weighting factor for pattern length l. For a given CDFG, the function f computes a value which is an estimate for the required number of connection paths of length l. For correctly adjusted coefficients a_l, the resulting cost value should deliver an estimate for the expected number of connections in the resulting circuit. To compute $f(\mathrm{cdfg}, l)$ an undirected graph G_l is constructed. The nodes of G_l are the patterns of length l. An edge in G_l denotes a match between the corresponding patterns. The number of cliques of G_l gives an estimate for the number of required connection paths of length l.

For the experiments described in section 6, we determined the coefficients $a_l, l = 2, 3, 4$ based on the results of synthesizing the benchmarks of table 1 several times with different hardware resources, each time computing f for each length. Afterwards an allocation was performed to determine the effective connection count r. Thus, each synthesis delivered an equation

$$a_2 f(cdfg, 2) + a_3 f(cdfg, 3) + a_4 f(cdfg, 4) = r$$

The set of equations was solved using the least-squares method. The results suggested that a ratio of about 9:3:1 for $a_2 : a_3 : a_4$ would be a reasonable choice for most cases.

5 Algorithm Outline

The algorithm consists of three phases that are performed iteratively until no further improvements can be achieved.

1. collect alignments: Collect all patterns of the CDFG up to length 4 that can be subject to alignment. Heuristics like critical path length are used to reduce the number of patterns to be considered for insertions. Based on these patterns a set of possible alignments is determined. To limit the complexity, only alignments containing up to two insertions are considered.

2. sort by cost: Sort the alignments according to their cost, using the cost function described in section 4.1. For efficiency reasons, the clique partitioning during calculation of the cost function is performed heuristically.

3. select: Starting with the best alignment, apply the insertions of the current alignment and synthesize the resulting CDFG to determine the interconnect cost. Synthesis is required to determine whether the inserted operations have lead to the expected interconnect savings. If no improvement is achieved, try the next best alignment. Otherwise, the current alignment is accepted and the resulting CDFG is taken as the basis for the next iteration.

6 Experiments

Table 2 shows the interconnect savings achieved by our approach for three different register models. For each register model, the column *before* holds the numbers of connections that were used without inserting any identity operations, the column *after* shows the numbers of connections after applying our algorithm and the column *gain* shows the difference in percent. The experiments

	single registers			single port register files			multi port register files		
	before	after	gain	before	after	gain	before	after	gain
binary search	12	10	16,7%	19	16	15,8%	9	8	11,1%
median filter	9	8	11,1%	11	9	18,2%	7	7	—
bubble sort	3	2	33,3%	12	10	16,7%	5	5	—
FIR filter	12	11	8,3%	25	22	12,0%	11	9	18,2%
blue screen	9	7	22,2%	13	9	30,8%	4	4	—

Table 2: Results: Interconnect Cost Savings

were performed for different sets of hardware resources (1–3 ALUs and 1–2 multipliers). We used a force directed list scheduler and simulated annealing for allocation.

The computation of the cost function for a single alignment takes less than one second on our system. In contrast, a full synthesis can take several minutes. The quality of the cost function therefore is crucial to avoid unnecessary iterations. For our examples, in 50% of the cases, the alignment with minimum estimated cost lead to a interconnect cost reduction. The average number of alignments that had to be tried until an improvement was achieved was 2.48.

The *single registers* model does not group registers into files. With this model, the number of connections is largest. However, if no registers are connected to other registers, for a given number of r registers the theoretical minimum for the number of connections is $2r$. Thus, table 2 shows only the numbers of connections that were used in addition to this theoretical minimum. The other two models use register files. The single port register files only allow a single read and write operation per cycle, thus requiring a larger number of register files than in case of the multi-ported register files. Again, only the number of connections in addition to the theoretical minima are shown. Since all three register file models differ in the way connections between FUs and registers are shared, the numbers of connections differ for all three models. The multi port register files offer the highest degree for interconnect sharing, which means that a single connection can correspond to several edges in the CDFG. Consequently, to eliminate such a connection, insertions for all of the corresponding edges have to be performed. Since we only considered alignments with up to two insertions, improvements are unlikely, as table 2 shows.

Highly regular algorithms like the FIR filter already offer high degree of potential interconnect sharing. There are only few patterns which don't match any others, which means there is only little potential for interconnect savings. But, since a single insertion can suffice to make an isolated pattern match with others, these few patterns are easily eliminated. Thus, even for multi port register files two connections could be saved for the FIR filter algorithm. In contrast, algorithms which perform irregular computations or expose a lot of control flow contain a larger variety of patterns which might indicate a larger potential for interconnect sharing. However, it showed to be relatively rare that a single insertion eliminates the need for a connection. Thus, there are only slight differences between the savings for the different algorithms.

In most cases the algorithm is able to reduce the number of connections above the theoretical minimum between 10% and 30%. For better results with irregular computations or with multi port registers, the set of considered alignments would have to be extended for alignments with more than two insertions. To cope with the additional complexity, this requires more elaborate heuristics for a better pre-selection of promising alignments.

7 Conclusion

A high level CDFG transformation increasing the potential for interconnect minimization was presented. By insertion of one or more identity operations, computation patterns are aligned to potentially share the same connections. Since the actual interconnect savings depend on the following synthesis steps of scheduling and allocation, a function selecting the suitable CDFG edges for identity operation insertion must be developed. Based on statistical experiments, we defined a linear cost function for transformation selection which was then applied to practical benchmarks on different register architectures. The benchmarks showed the suitability of the approach leading to typical interconnect overhead reduction between 10% and 30%.

The approach is easily integrated into the common high-level synthesis design flow, either as a separate optimization step or in interaction with other synthesis phases. It is independent of the hardware models like the register file structure.

References

[1] Michael R. Garey and David S. Johnson. *Computers and intractability: a guide to the theory of NP-completeness*. A series of books in the mathematical science. Freeman, New York, 1979.

[2] W. Geurts, F. Catthoor, and H. DeMan. Quadratic zero-one programming-based synthesis of application-specific data paths. *IEEE Trans. on CAD*, 14(1):1–11, January 1995.

[3] B. Landwehr and P. Marwedel. A new optimization technique for improving resource exploitation and critical path minimization. In *ISSS*, pages 65–72, 1997.

[4] D. Lobo and B. Pangrle. Redundant operator creation: A scheduling optimization technique. In *DAC*, page 775, 1991.

[5] N. Park and F. J. Kurdahi. Module assignment and interconnect sharing in register-transfer synthesis of pipelined data paths. In *ICCAD*, pages 16–19, 1989.

[6] M. Potkonjak and S. Dey. Optimizing resource utilization and testability using hot potato techniques. In *DAC*, pages 201–206, 1994.

[7] W. Verhaegh, M. Peek, P. Lippens, E. Aarts, A. van der Werft, and J. van Meerbergen. Area optimization of multi-functional processing units. In *ICCAD*, page 292, 1992.

[8] D. Ziegenbein. Analysis of an approach to reduce interconnect structures by insertion of redundant operations in high-level synthesis (in german). Master's thesis, Technische Universität Braunschweig, 1997.